Vysoká škola báňská – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Jednoduchý herní engine pro Android

Simple Game Engine for Android

2014 Michal Zich

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Michal Zich**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Jednoduchý herní engine pro Android
Simple Game Engine for Android

Zásady pro vypracování:

Cílem bakalářské práce je implementace jednoduchého herního enginu pro platformu Android. Engine umožní vytvářet a zobrazovat objekty, jejich pohyb a jednoduché interakce mezi definovanými objekty ve dvourozměrném prostoru.

Úkoly:
1. Analýza platformy Android a jejích možností.
2. Rešerše existujících řešení.
3. Návrh a implementace vlastního enginu.
4. Vytvoření ukázkových aplikací, srovnání s existujícími rámci.

Seznam doporučené odborné literatury:

ZECHNER, Mario. Beginning Android games. New York: Apress, c2011, xvi, 669 s. ISBN 978-1-4302-3042-7

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Roman Meca**

Datum zadání: 01.09.2013
Datum odevzdání: 07.05.2014

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * OSTRAVA *

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 2.5.2014

Michael Zich

.....

Rád bych na tomto místě poděkoval vedoucímu práce, panu Ing. Romanu Mecovi, který mě svými radami vedl tím správným směrem.

Abstrakt

Cílem bakalářské práce je implementace jednoduchého herního enginu pro platformu Android. Engine umožní vytvářet a zobrazovat objekty, jejich pohyb a jednoduché interakce mezi definovanými objekty ve dvourozměrném prostoru. V první části bude čtenář seznámen s platformou Android a najde zde základní informace. V druhé části budou popsány již vytvořené enginy, které mohou být použity při tvorbě her. Třetí část bakalářské práce bude věnována praktické části, ve které bude vytvořen jednoduchý engine pomocí jazyka Java a nástroje pro vykreslování grafiky OpenGL ES. Pomocí tohoto enginu budou v poslední části vytvořeny jednoduché hry.

Klíčová slova: Android, herní engine, bakalářská práce, OpenGL ES, Java, Eclipse, Google Play

Abstract

The main aim of the bachelor thesis is the implementation of simple game engine for Android platform. Engine provides creating and displaying objects, their movement and simple interactions between defined objects in two-dimensional space. In the first part, the Android platform and basic information will be introduced. The second part will describe already formed engines that may be used in the creation of the games. The following section will deal with the practical part of the bachelor thesis in which a simple engine will be created by using Java and OpenGL ES, a tool for graphic rendering. In the last part of the bachelor thesis, several simple games will be created by using this engine.

Keywords: Android, Game Engine, bachelor thesis, OpenGL ES, Java, Eclipse, Google Play

Seznam použitých zkratek a symbolů

APK	–	Android application package file
Bitmapa	–	Rastrový obrázek, resp. jeho reprezentace v paměti počítače jako pole pixelů
Box2D	–	Open source 2D engine napsaný v C++
CAD	–	Computer-Aided Design - počítačem podporavané projektování
DP	–	Display Picture
FPS	–	First Person Shooting
HDML	–	Handheld Device Markup Language
IDE	–	Integrated Developement Enviroment
JIT	–	Just-in-time
Multiplayer	–	Možnost hraní her pro více hráčů
MP3	–	MPEG-1 nebo MPEG-2 Audio Layer III
NDK	–	Native Developement Kit
Open Source	–	Kód volně ke stažení s možností úprav a rozšiřování
OpenGL ES	–	OpenGL for Embedded Systems
RPG	–	Role-playing game
RTS	–	Real-time strategy
SDK	–	Software Developement Kit
Sprite	–	Malý dvojrozměrný obrázek, který je integrován do větší scény
Startup	–	Nově vznikající projekt či začínající firma často ještě ve fázi tvorby podnikatelského záměru
OS	–	Operační systém
OHA	–	Open Handset Alliance

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

Úvod

Platforma Android je téměř všude. Najdete ho v mobilních telefonech, tabletech, televizích a set-top boxech s technologií Google TV a brzy bude i v automobilech, zábavních systémech letadel, a taktéž i v robotech.

Hlavní oblastí uplatnění systému Android však budou i nadále zařízení s menšími obrazovkami a s hardwarovou klávesnicí či bez ní. Ze statistického hlediska bude pak systém Android v blízké budoucnosti pravděpodobné i nadále spojován především s chytrými telefony [1].

Aplikaci pro mobilní zařízení dnes nenabízí jenom nejrůznější webové služby, ale i operátoři, banky a zpravodajské weby. Vlastní aplikaci se zajímavými informacemi mívají i velké veletrhy a sportovní akce (například olympiáda). Zařízení s Androidem, ať už se jedná o telefony, nebo tablety, se těší velké oblibě. Nabídka je značně široká, uživatel si může vybírat z několika desítek zařízení nejrůznější velikostí, výkonů a také cen. Cena nejlevnějších telefonů se pohybuje kolem třech tisíc korun českých, vstup do světa systému Android tedy není ani zdaleka tak nákladný jako na konkurenční platformě Apple.

Téma bakalářské práce jsem si zvolil hlavně díky rozšířenosti platformy Android a přesvědčení, že má tento operační systém velmi důležité místo ve společnosti, ale i přesto je mnoha lidmi odsuzován. Již dříve jsem se věnoval tvorbě jednoduchých aplikací pro Android a hlavní motivací pro mě bylo zdokonailt se v oboru, kterým se chci i nadále zabývat.

Cílem této bakalářské práce bude vývoj jednoduchého herního enginu pro platformu Android od implementace samotného enginu po tvorbu her na mnou vytvořený engine. První část práce bude zaměřena na informace o samotném systému Android. Bude zde zmíněna jak historie a současnost platformy, tak výhody a nevýhody programování aplikací pro tento operační systém. Čtenář zde najde typy programovacích jazyků, které lze při tvorbě využít, ale i samotné části, ze kterých se Android skládá. Druhá část práce se bude zabývat obecnými informacemi o již existujících enginech, které mohou být použity při tvorbě her. Třetí část bakalářské práce bude věnována praktické části, ve které bude vytvořen jednoduchý engine pomocí jazyka Java a nástroje pro vykreslování grafiky OpenGL ES, který je určen pro mobilní platformy. Pomocí tohoto enginu bude v poslední části vytvořeno více her, což bude znázorňovat jeho opětovnou použitelnost.

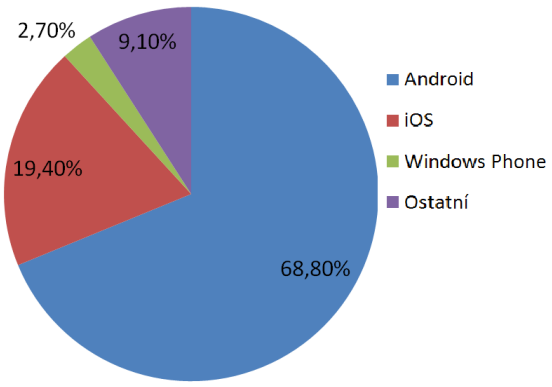
1 Analýza platformy Android a jejich možností

Kapitola seznámí čtenáře se samotnou platformou Android. Lze zde nalézt zmínky z historie i současnosti, ale i výhody a nevýhody programování pro tento operační systém. Kapitola umožňuje udělat si přehled o základních pojmech souvisejících právě s Androidem.

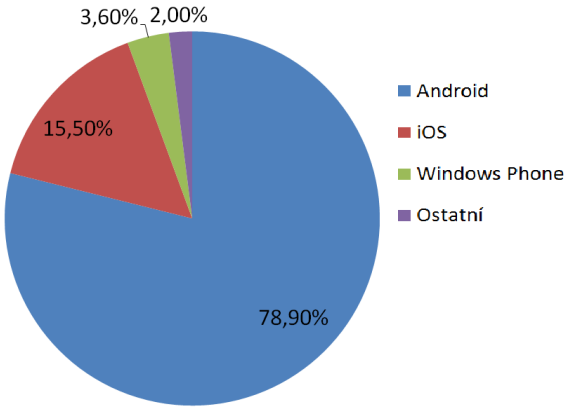
1.1 Seznámení s platformou Android

Ne všichni lidé vědí, že Android nepochází původně od společnosti Google. V roce 2003 založili pánové Andy Rubin, Rich Miner, Nick Sears a Chris White společnost Android Inc. V srpnu roku 2005 Google Inc. odkoupil začínající startup a udělal z něj svou dceřinou společnost. V roce 2007 získal Google patenty na mobilní operační systém a plánuje vstoupit na trh se svým vlastním operačním systémem pro mobilní telefony. V listopadu téhož roku bylo vytvořeno seskupení OHA, jehož cílem bylo vytvořit otevřený standard pro mobilní telefony. Trvalo téměř rok, než byl vydán první telefon značky HTC s Androidem SDK 1.0. Na český trh se dostal počátkem roku 2009.

Architektura operačního systému Android obsahuje Linuxové jádro ve verzi 2.6., přičemž je využito mnoha jeho vlastností, jako jsou podpora správy paměti, správa sítí, správy procesů. Tyto vlastnosti přispívají ke stabilitě systému. Grafy 1.1 a 1.2 zobrazují podíly operačních systémů na trhu za roky 2012 a 2013, kde systém Android dominuje s téměř 80 procenty k roku 2013 [2].



Obrázek 1.1: Podíl OS za rok 2012

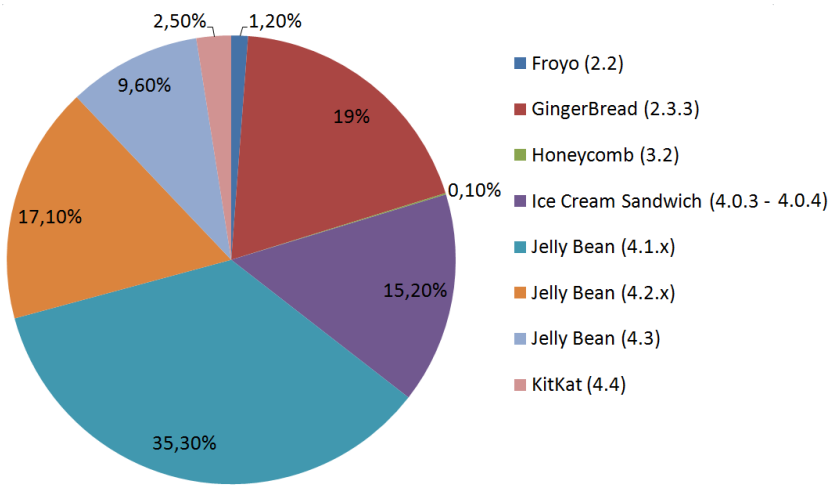


Obrázek 1.2: Podíl OS za rok 2013

1.2 Verze Androidu

Jeden z půvabů vývoje pro platformu Android spočívá v tom, že se široce používá v mnoha různých zařízeních, jako jsou mobilní telefony, tablety, přehrávače MP3 a čtečky knih. V celém spektru nejrůznější výrobců se hardware, na němž může být aplikace nainstalována, docela liší.

Na tuto okolnost lze pohlížet také jako na překážku, kterou je nutno překročit. V kterémkoli okamžiku by totiž mohlo existovat, v době psaní práce až 19, odlišných verzí platformy Android fungujících na odlišných verzích hardwaru. Verze byly pojmenovány po laskominách a od první verze bylo vydáno několik aktualizací. Aktualizace opravují chyby a přidávají nové funkce. Na nejnovějších tabletech a telefonech jsou nainstalovány verze 4.0 (Ice Cream Sandwich), 4.1 (Jelly Bean) nebo 4.4 (Kit Kat) - což je nejnovější verze. Procentuální využití verzí Androidu možno vidět v grafu 1.3. Data jsou aktuální ke konci března roku 2014.



Obrázek 1.3: Procentuální využití verzí Android k březnu 2014 [3].

1.3 Výhody a nevýhody programování pro chytré telefony

Hlavní výhodou zařízení obsahující systém Android je jejich přitažlivost. Možnost používání internetových služeb v mobilních zařízeních je zde již od poloviny devadesátých let a vzniku jazyka HDML. Opravdový rozmach telefonů umožňujících přístup k Internetu však nastal teprve v posledních letech. Díky trendům, jako je zasílání online textových zpráv nebo zařízení iPhone společnosti Apple, popularita telefonů, které lze použít jako přístupový bod k Internetu, rychle vzrůstá.

Problém nastává až při samotném programování aplikací. Při programování aplikací pro chytré telefony nastávají nepříjemnosti plynoucí z prostého faktu, že jsou telefony ve všech směrech jednoduše velmi malé. Mezi časté problémy chytrých telefonů patří:

- malé obrazovky,
- malé klávesnice,
- nepraktická polohovací zařízení (pokud nějaká mají), nepřesná (uživatelé s velkými prsty často mívají potíže s ovládáním vícedotkových obrazovek).

1.4 Čím je systém Android tvořen

Systém Android využívá podobné koncepce jako při programování aplikací pro počítače. Jsou však jinak sdruženy a strukturovány, aby byly telefony odolnější vůči chybám.

Aktivity: stavební kameny pro tvorbu uživatelského rozhraní. Může být chápáno jako analogie oken či dialogů klasické aplikace pro počítače nebo jako stránka klasické webové aplikace. Systém Android je navržen tak, aby podporoval množství nenáročných aktivit. Uživatel může nové aktivity klepnutím na tlačítko otevírat a navrátit se do dříve otevřené aktivity pomocí tlačítka zpět stejně jako ve webovém prohlížeči. Aktivity jsou entity s krátkým životním cyklem a lze je kdykoli ukončit.

Služby: služby jsou navrženy k neustálému provozu, a pokud je potřeba, aby pokračovaly ve své práci nezávisle na jakékoli aktivitě. Služby lze použít například k přehrávání hudby na pozadí. Hudba může hrát dál, i když je aplikace zavřena (aktivita neběží).

Poskytovatelé obsahu: zajišťují úroveň abstrakce jakýchkoli dat uložených v zařízení, která jsou přípustná více různým aplikacím. Současně poskytují úplnou kontrolu nad způsobem přístupu k datům uživatele. Lze je kdykoli vypnout. Je to jediná cesta sdílení dat napříč aplikacemi. Poskytovatelem obsahu může být cokoli, od webových kanálů po místní databází SQLite.

Záměry: systémové zprávy upozorňující aplikace na výskyt různých událostí změnami hardwarové konfigurace (například vložení SD karty) počínaje, přes události související s příchozími daty (například přijetí zprávy SMS), po události aplikací (například spuštění aktivity) konče. Na záměry můžeme reagovat. Ale i vytvářet vlastní a spouštět jimi jiné aktivity nebo se nechat informovat o určitých situacích.

Navíc rozlišujeme 4 typy aplikací:

- foreground - v popředí - použitelné jen v popředí - pozastavené, když jsou neviditelné,
- background - v pozadí - limitovaná interakce, obvykle jen konfigurace,
- intermittent - přerušované – například přehrávač médií, pracují v pozadí,
- widget - viditelné jen na domácí ploše, například počasí nebo indikátor baterie.

1.4.1 Životní cyklus aktivity

Životní cyklus aktivity zásadním způsobem ovlivňuje podobu její aplikační logiky. Každá aktivita se obecně vzato vždy nachází v jednom z následujících 4 stavů:

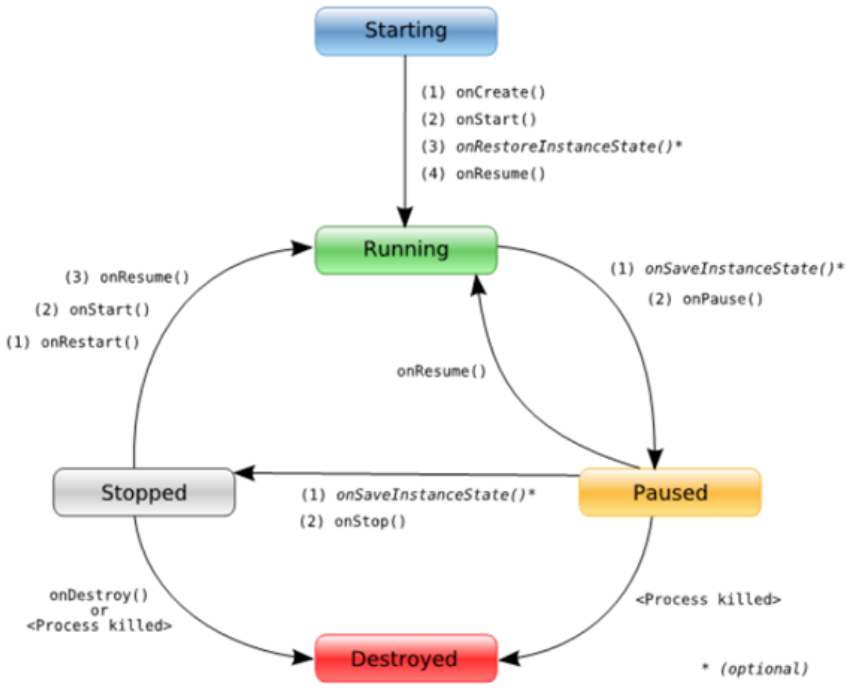
Aktivní: Aktivitu spustil uživatel a běží v popředí. Takto si obvykle chod aktivity představujeme.

Pozastavená: Aktivitu spustil uživatel, běží a je viditelná, ale část její obrazovky překrývá nějaké upozornění. Uživatel aktivitu vidí, ale nemůže s ní nijak interagovat. Dochází k tomu v případě, že máme příchozí hovor.

Zastavená: Aktivitu spustil uživatel, běží, ale je skrytá za jinými aktivitami, které byly spuštěny po ní nebo se aktivovaly. Aktivita nemůže nic přímo zobrazit, ale může s uživatelem komunikovat pomocí upozornění (notifications).

Mrtvá: Uživatel aktivitu buď vůbec nespustil (došlo k restartu telefonu), nebo byla násilně ukončena (například kvůli nedostatku paměti).

Obrázek 1.4 znázorňuje životní cyklus aktivity.



Obrázek 1.4: Životní cyklus aktivity [5].

1.5 Programování aplikací pro Android

Vyvíjení aplikací na platformu Android má své výhody i nevýhody, kterých by si měl být programátor vědom ještě před tím, než začne programovat. Aplikace mohou být vyvíjeny v jazyce Java. Zdrojový kód je následně převeden do formátu, se kterým systém Android skutečně pracuje (do kódu bytecode Dalvik zabaleného do souboru balíčku systému Android - souboru APK). Android však není kompletní implementací Javy. Spousta balíčků, používaných pro hry, jako je OpenGL a další grafické dekorace, je obsažená v sadě SKD platformy Android. Ovšem některé skutečně užitečné balíčky pro vývojáře her, zvláště pak pro vývojáře trojrozměrných her, zde obsaženy nejsou.

Sada Android SDK poskytuje nástroje potřebné k vytváření a testování aplikací pro Android. Sada se skládá ze dvou částí: první z nich tvoří základní nástroje, druhou komponenty specifické pro konkrétní verze a příbuzné doplňky. S každým vydáním nové sady Android SDK je k dispozici více a více balíčků a starší mohou být označeny jako zastaralé. Programátor si tedy musí dát pozor, s kterými balíčky pracuje.

1.6 Definice rozložení ve formátu XML

Ačkoli je technicky vzato možné vytvářet a připojovat widgety k aktivitám pouze prostřednictvím zdrojového kódu jazyka Java, obvyklejší způsob spočívá v použití souboru definice rozložení ve formátu XML.

Definice rozložení ve formátu XML je specifikace vzájemných vztahů widgetů a jejich vztahů k obsahujícímu kontejneru ve formátu XML. Každý soubor XML obsahuje strom elementů, který definuje rozložení widgetů a kontejnerů, které tvoří jednu instanci třídy View. Atributy jednotlivých elementů XML jsou vlastnosti popisující vzhled widgetu nebo chování kontejneru.

Hlavní výhodou rozložení ve formátu XML je přenositelnost mezi verzemi. To, co je naprogramováno pro danou aplikaci, bude vypadat stejně na zařízeních s verzí systému 2.3.3 i například s verzemi 4.0.

1.7 Android Manifest

Základem každé aplikace pro Android je soubor manifestu AndroidManifest.xml, který je uložen v kořenovém adresáři projektu. V tomto souboru je deklarován obsah aplikace - aktivity, služby apod. Zde je také uvedeno, jak jsou tyto součásti aplikace propojeny s operačním systémem - například která aktivita se má zobrazit v hlavní nabídce zařízení. Většinou nastává problém se spuštěním, když není aktivita přidána v manifestu.

Při programování jednoduché aplikace s jednou aktivitou není potřeba manifest nijak upravovat. Avšak při distribuci aplikace je potřeba soubor upravit. Manifest určuje hodnoty atributů android:versionName a android:versionCode, jejichž hodnoty reprezentují verzi aplikace a jsou podmínkou při produkování aplikace na Google Play. Android i služba Google Play porovnává kód verze nového souboru APK s kódem verze nainstalované aplikace, aby určily, zda je nový soubor APK opravdu novější než stávající. Soubor AndroidManifest.xml může obsahovat i následující elementy:

- uses-permission - práva pro správný chod aplikace,
- permission - práva udělující pro jiné aplikace, aby mohly používat logiku mé aplikace,
- uses-sdk - určující verzi systému podporující danou aplikaci, například pomocí parametru android:minSdkVersion určuje nejstarší použitelnou verzi,
- supports-screens - uvádí všechny velikosti obrazovky, které aplikace explicitně povoluje,
- a mnoho dalších.

1.8 A nejen Java

Možná si nyní myslíte, že aplikace pro Android lze programovat pouze v jazyce Java. Oficiální sada SDK pro systém Android je opravdu určena k vývoji pomocí jazyka Java

a většina knih související s problematikou programování pro Android je také věnována jazyku Java. Pro tuto práci byl také zvolen jazyk Java kvůli bohaté dokumentaci. To však neznamená, že nelze programovat i v jiném jazyce.

I Když programátor programuje aplikace v jazyce Java, zařízení se systémem Android ve skutečnosti aplikaci před spuštěním konvertuje na kód, který následně provádí virtuální stroj Dalvik. Jedná se o podobnou technologii, která se používá v případě běžných aplikací naprogramovaných v jazyce Java, ale virtuální stroj Dalvik je speciálně vyladěn k použití v prostředí systému Android. Navíc omezuje závislost systému Android na jazyce Java. Virtuální stroj Dalvik umožňuje spouštění kódů jiných programovacích jazyků.

Na počátku se při interpretaci zdrojového kódu jazyka Java v jeho výchozí implementaci pro virtuální stroj Dalvik nepoužívaly žádné kompilační techniky JIT, které běžné interprety jazyka Java používají ke zvýšení výkonu. Tento fakt představoval problém. Zejména s ohledem na to, že zařízení vybavená systémem Android obvykle nedisponují takovými výpočetními prostředky jako dnešní počítače či notebooky. Ve verzi systému Android 2.3 byl proto přidán kompilátor JIT, který tuto situaci výrazně zlepšil, ale který stále není schopen dosáhnout výkonu nativního zkompilovaného kódu. Proto se programátor může setkat při programování některých úkolů s problémy, které pomocí jazyka Java nevyřeší z toho důvodu, že je jeho zdrojový kód příliš pomalý.

Nyní mám na mysli například programování složitých 3D her, pro které se používají jazyky jako C nebo C++ se sadou Android NDK. NDK je sada nástrojů, která umožňuje programátorovi implementovat části aplikace pomocí nativního kódu programovanými jazyky jako například C nebo C++. U některých typů aplikací, právě u her, může být použití užitečné. Programátor může znovu použít existující kód knihovny napsané v těchto jazycích. Většina aplikací ale Android NDK nepotřebuje. Avšak při použití může být kód až několikanásobně rychlejší než při programování v jazyku Java. Například algoritmy pro obsluhu fyzikálních prostředí, jejichž přeprogramování v Javě by bylo velmi náročné.

Za zmínku stojí i programování aplikací pomocí HTML5. Obecně vzato jazyky C/C++ jsou výkonnější než jazyk Java, ale jazyk HTML5 již nikoliv. V závislosti na tom, co je programováno a do jaké míry je výsledná aplikace využívána, pak lze určit, jak se tato neefektivita projeví.

1.9 Hardware a Android

Telefonů a tabletů se systémem Android je obrovská škála. Od levných tabletů za 2000 Kč po výkonné tablety přesahující cenu 10 tisíc Kč, na kterých si uživatel zahraje i ty nejlepší a nejnáročnější hry. Ovšem v dnešní době se dá pořídit kvalitní tablet na hraní her již za 5 tisíc Kč a může obsahovat čtyřjádrový procesor. Na čínském trhu se objevují dokonce telefony s výkonnými 8 jádrovými procesory.

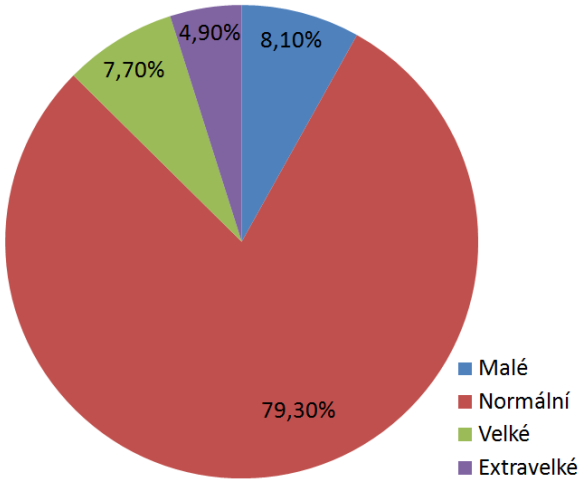
Konkurenční platforma Apple vyrábí dva typy tabletů, které jsou výkonné, a většina aplikací na nich běží plynule, ale cena je vyšší než u srovnatelných tabletů se systémem Android. U testovacího hardwaru Sony Xpreria T, který obsahuje dvoujádrový procesor Qualcomm Snapdragon S4, s frekvencí procesoru 1,5 GHz a operační pamětí RAM 1 GB, funguje většina, i náročnějších her, zcela bez problémů.

1.9.1 Podpora různých displejů obrazovky

Zařízení se vyrábějí s celou řadou různých rozměrů obrazovek, od maličkých chytrých telefonů s obrazovkou o úhlopříčce okolo 7cm až po Televize Google s obrazovkou o úhlopříčce okolo 120cm. Což je pro programátora nevýhoda. Musí myslet na všechny typy obrazovek a ladit svou aplikaci na velkou škálu displejů. Oproti konkurenční platformě Apple, která má pouze 4 rozměry displejů. Systém Android rozděluje obrazovky zařízení do čtyř kategorií na základě jejich rozměrů a vzdálenosti, ze které jsou obvykle pozorovány:

- malé: s úhlopříčkou pod 7,5 cm a rozlišením alespoň 426 x 320 dp,
- normální: s úhlopříčkou od 7,5 cm po zhruba 11,5 cm a rozlišením alespoň 470 x 320 dp,
- velké: s úhlopříčkou od 11,5 cm po zhruba 25 cm a rozlišením alespoň 640 x 480 dp,
- extravelké: s úhlopříčkou nad 25 cm a rozlišením alespoň 960 x 720 dp.

Aplikace ve výchozím nastavení nepodporují malé obrazovky, podporují normální obrazovky a pomocí kódu systému Android k automatické konverzi, škálování a změnám velikosti mohou podporovat velké i extra velké obrazovky. Graf 1.5 znázorňuje údaje o relativním počtu zařízení, která mají konkrétní konfiguraci obrazovky.



Obrázek 1.5: Graf znázorňující údaje o relativním počtu zařízení, která mají konkrétní konfiguraci obrazovky [3].

1.10 Trh s aplikacemi

Většinu aplikací, o které má uživatel zájem, je možno najít v Google Play. Google Play je služba, která zpřístupňuje aplikace vytvořené vývojáři po celém světě. Takže jakýkoli

uživatel si může stáhnout kteroukoli aplikaci umístěnou na tomto portále za předpokladu, že byla povolena pro zemi, ve které se nachází. V případě placené aplikace musí být země nákupu dostupná v seznamu povolených států (ne všechny země mají povolenou distribuci a koupi placených aplikací). Existují však i alternativní obchody. Kvůli zmíněné nevýhodě s distribucí v omezení obsahu pro určitá zařízení či určité země se může stát, že Google Play nepovolí stáhnout aplikaci či hru, i přes to, že na daném telefonu funguje. Mezi alternativní obchody patří například Amazon Appstore, F-Droid, SlideMe, 1Mobile a mnoho dalších. Za zmínku stojí Amazon Appstore, který nabízí každý den jednu aplikaci či hru zdarma [4]. Aplikace se dělí do těchto kategorií:

- zdarma - aplikace jsou ke stažení zdarma,
- zpoplatněné - za aplikace musí uživatel zaplatit částku stanovenou vývojářem či prodejcem,
- freemium bussines model - aplikace je zdarma, ale pro lepší funkce je potřeba si připlatit nebo přikoupit vylepšení. Do tohoto modelu patří aplikace, které na sebe vydělávají pomocí reklamy, která je zobrazena v aplikaci. Patří sem i tzv. mikro transakce - aplikace, především hry, jsou ke stažení zdarma. Hráč si však může koupit například nové zbraně či levely za peníze, aniž by musel hru hrát hodiny a k výsledku se sám dopracoval. Některé hry povolují vyšší levely pouze po zaplacení dané částky.

Pro vstup na trh jakožto vývojář není nijak složité. Stačí zaplatit jednorázový poplatek 25 dolarů. Zatímco u platformy Android stačí tento poplatek zaplatit jednou, u konkurenční platformy Apple se musí platit každý rok a cena činí 99 dolarů. Každý uživatel má po stažení možnost ohodnotit aplikaci jak slovy, tak počtem hvězd v rozmezí 1 až 5, kde 5 je nejlepší možnost. Tato odezva může pomoci vývojářům upravovat své aplikace ke spokojenosti zákazníků. Vývojář po přihlášení na web Google Play vidí statistiky své aplikace. Ty zahrnují například přesný počet stažení, pády aplikace či počet hodnocení.

2 Rešerše existujících řešení

V srdci každé videohry je herní engine (motor hry). Jak již jeho název napovídá, herní engine je kód, který pohání hru. Každá hra, bez ohledu na svůj typ (RPG, FPS, skákačka nebo RTS) vyžaduje pro svůj běh nějaký engine. Engine jakékoli hry je záměrně sestaven tak, aby byl generický a umožňoval své použití ve více situacích a i více různých hrách. To je v přímém protikladu ke kódu specifickému pro hru, který, jak již jeho název napovídá, je specifický pouze pro jedinou hru. Všeobecně lze říci, že herní engine se stará o veškerou rutinní práci herního kódu. To může znamenat cokoliv, od přehrání zvuku po vykreslení grafiky na obrazovku. Tato kapitola obsahuje již existující enginy, pomocí kterých mohou vývojáři tvořit hry. Zde je krátký seznam funkcí, které provádí typický herní engine:

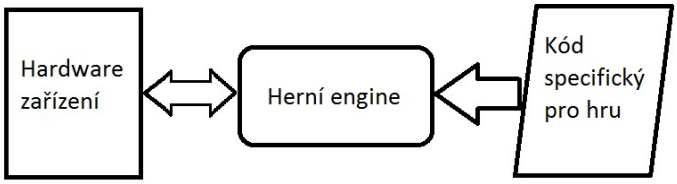
- vykreslování grafiky,
- animace,
- zvuk,
- detekce kolizí,
- umělá inteligence,
- fyzika,
- správa vláken a paměti,
- komunikace přes síť,
- interpret příkazů (vstup a výstup).

Herní engine slouží k tomu, aby hra fungovala efektivně, nemůžeme však spoléhat na to, že by tento druh zátěžové práce prováděl operační systém hostitelského systému. Většina operačních systémů má vestavěné prvky, které se dokáží postarat o každou položku z tohoto seznamu. Nicméně tyto systémy operačního systému pro vykreslování, zvuk a správu paměti jsou sestavené tak, aby na nich fungoval operační systém a aby se dokázaly přizpůsobit jakémukoli počtu nepředvídatelných použití, aniž by se na jakýkoli z nich specializovaly. Hry vyžadují něco s trošku větším výkonem.

K tomu, aby byla hra rychlá a plynulá, musí kód obejít zátěž, kterou standardní operační systémy vytvářejí, a běžet přímo nad hardwarem vyžadovaným pro specifický proces. To znamená, že hra by měla komunikovat pro provádění grafických funkcí přímo s grafickým hardwarem, pro přehrávání efektů přímo se zvukovou kartou a tak dále. Pokud by byly použity standardní paměťové, grafické a zvukové systémy, které jsou dostupné u většiny operačních systémů, vlákna hry by se chovala stejně jako vlákna ostatních funkcí operačního systému a aplikací běžících na tomto systému. Interní zprávy by se navíc mohly řadit do fronty s jakoukoli další systémovou zprávou. To by způsobilo, že by hra byla pomalá a vypadala trhaně.

Z tohoto důvodu se herní enginy téměř vždy programují v jazycích nízké úrovně. Jazyky nízké úrovně nabízejí přímější cestu k hardwaru systému. Herní engine musí být

schopen pracovat s kódem a s příkazy z kódu specifického pro hru a předat je přímo určitému hardwaru. Díky tomu může hra fungovat mnohem rychleji a s veškerou kontrolou, kterou potřebuje k tomu, aby poskytovala uspokojující zážitek. Obrázek 2.1 znázorňuje zjednodušenou verzi vztahu mezi herním enginem, hardwarem zařízení a kódem specifickým pro hru.



Obrázek 2.1: Vztah mezi herním enginem, kódem specifickým pro hru a hardwarem zařízení [15].

Herní engine nedělá nic speciálně pro určitou hru. To znamená, že herní engine nevykresluje na obrazovku raketu. Herní engine na obrazovku něco vykreslí, protože obstarává vykreslování grafiky, sám od sebe ale nic určitého nevykresluje. Je úkolem kódu specifického pro hru, aby předal enginu raketu, která má být vykreslena, a je úkolem enginu, aby vykreslil cokoliv, co mu je předáno.

Podstatné je, že engine je velice obecný, zatímco kód specifický pro hru nikoliv. U větších projektů, jako je třeba hra, na níž pracují desítky nebo stovky lidí, se nejdříve vyvine engine a potom se vytvoří kód specifický pro hru tak, aby pracoval s tímto enginem. V případě malých her, jako jsou ty, které jsou vytvářeny v rámci bakalářské práce, je možné současně vyvíjet herní engine a kód specifický pro hru.

2.1 Unreal Engine

Jedním z velice oblíbených herních enginů je Unreal Engine. Unreal Engine, poprvé vyvinutý roku 1998 společností Epic Games pro její střílečku z vlastního pohledu s názvem Unreal, byl nasazený ve stovkách her. Unreal Engine lze snadno adaptovat a pracuje s nejrůznějšími typy her, ne jen s hrami typu FPS. Díky této generické struktuře a flexibilitě je oblíbený nejen mezi profesionály, ale také příležitostnými vývojáři.

Jádro enginu je napsáno v jazyce C++ a je podporováno na mnoha platformách, jako jsou Microsoft Windows, Linux, Mac OS i Android. Lze tento engine vidět i na herních konzolích Xbox a PlayStation. Většina součástí tohoto enginu je napsáno ve speciálním kódu zvaném UnrealScript. Proto není nutné při vytváření herních módů přímo zasahovat do jádra, ale stačí pouze měnit určité scripty.

Nejnovější verzí je Unreal Engine 4, jejíž vývoj začal již v roce 2003 a byl představen až v roce 2012. Společnost Epic Games nabízí engine za poplatek ve výši 19 dolarů měsíčně a pět procent z výnosu. Za předplatné je možné získat zdrojový kód v jazyce C++ a kompletní sadu vývojářských nástrojů, kterou využívá i Epic Games k tvorbě svých vlastních titulů. Společnost na situaci reagovala proto, že licencování přední technologie

Unreal Engine 3 vyšlo na miliony dolarů. Pro většinu nezávislých tvůrců tak její využití nepřipadalo v potaz. K dispozici však byla alternativa v podobě nástrojů UDK, které bylo možné využívat za odvádění 25 % z výnosů nad milion korun [6].

Na tomto enginu bylo postaveno mnoho známých i méně známých her. K těm nejznámějším patří Duke Nukem Forever, hry o Harry Potterovi, BioShock 1 i 2, Lineage II, Ragnarok Online 2, Unreal Tournament, Medal of Honor a spousty dalších.

2.1.1 Unreal Engine na Android zařízení

Společnost Epic Games vydala demo aplikaci Epic Citadel postavenou na známém Unreal Engine 3. Kromě toho, že demo naznačuje technické možnosti přenosných zařízení, je obohaceno také o srovnávací měřítko, díky kterému si může hráč o výkonu zařízení udělat ještě lepší přehled.

Již před dlouhou dobou byla pro přístroje se systémem Apple iOS uvolněna aplikace Epic Citadel demonstrující možnosti Unreal Engine 3 na přenosných zařízeních. Je zde možnost procházení prostředí ze známé hry Infinity Blade. Prohlídka prostředí probíhá poklepáním na displej pro pohyb a tažením po displeji pro rozhlížení po scéně. Díky tomu může uživatel naplno docenit krásně graficky zpracované lokace [7].

2.2 Unity3D

Unity3D je multiplatformní herní engine s vlastním vývojovým prostředím. Unity umožňuje programovat hry pro webový prohlížeč, stolní počítače, herní konzole a mobilní zařízení. Vývojové prostředí se samo postará o zkompilování na tu plaftormu, na které si vývojář přeje svou hru publikovat.

Nejnovější verze má číslo 4.3.3 a zahrnuje podporu publikování na platformách iOS, Android, Windows Store, Windows Phone, BlackBerry, operační systémy na stolních počítačích a webové prohlížeče. Ve zkušební, 30 denní verzi, podporuje platformy iOS, Android, Windows Store, Windows Phone and BlackBerry. Kompletní verze enginu i s vývojovým prostředím stojí 1500 dolarů jednorázově nebo 75 dolarový poplatek každý měsíc. 30 denní zkušební verze je zdarma [8].

2.3 AndEngine

AndEngine je open source 2D OpenGL herní engine pro platformu Android. Engine byl vytvořen vývojářem Nicolasem Gramlichem a je zdarma ke stažení. Tento engine slouží pro vývoj jednoduchých 2D her a podporuje například použití Box2D fyziky či multiplayer. Je oblíbený u nezávislých nekomerčních vývojářů. K dispozici je poměrně aktivní fórum komunity. Kód sám o sobě je dobře navržen a je rozšiřován [9].

2.4 Corona

Corona je SDK vytvořen Walterem Luhem. Slouží programátorům softwaru pro tvorbu mobilních aplikace pro platformu iOS a Android. Corona používá jazyk C++, Lua,

OpenGL a další pro vytvoření grafických aplikací. Mimo obyčejné aplikace slouží i k tvorbě her. SDK lze zakoupit ve verzi Basic za 16 dolarů či verzi Pro za 49 dolarů měsíčně [10].

2.5 Cocos2d-x

Cocos2d je multiplatformní open source rámec. Je používán k tvorbě her, aplikací a uživatelského rozhraní napříč operačními systémy. Pro platformu Android je verze Cocos2d-x. Na platformě Android slouží k tvorbě her fungujících i na zařízeních se slabším hardwarem. Engine je ke stažení zdarma a je k dispozici dobrá dokumentace a podpora od rozšířené komunity.

Engine je používán jak jednotlivci, tak velkými společnostmi jako jsou Zynga, Wooga, Disney Mobile a další. K září roku 2013 dosáhlo Cocos2d-x 1.5 miliardy stažení. K mnohým z nich došlo pomocí služeb Goole Play a AppStore (obchod s aplikacemi u konkurenční platformy Apple) [11].

2.6 LibGDX

LibGDX je také multiplatformní open source engine ke stažení zdarma. Lze hry publikovat pro Windows, Mac, Linux, Android, iOS, BlackBerry a HTML5, všechny pomocí stejného kódu. Podporován komunitou může programátor programovat 2D nebo 3D hry na nízkých detailech. Ke studiu slouží dokumentace a tutoriály. Kód je napsán v jazyce Java s C a C++ komponentami [12].

3 Návrh a implementace vlastního enginu

Hlavní kapitolou bakalářské práce je samotná tvorba enginu. Byla zde použita grafická knihovna OpenGL ES, se kterou jsem se dříve nesetkal a naučil se s ní pracovat. Veškerá práce byla prováděna ve vývojovém prostředí Eclipse za použití sady Android SDK a programovacího jazyka Java.

3.1 Příběh hry

Každá hra, od nejjednodušší arkádové hry pro nejkomplexnější hry typu RPG, začíná příběhem. Příběhem nemusí být nic více nežli věta, například: představme si, že máme obrovskou vesmírnou loď, která dokáže střílet. Příběh ovšem může být i dlouhý jako kniha, a popisovat každé území, osobu a zvíře v prostředí hry. Mohl by dokonce popisovat i každou zbraň, úkol a výsledek. Mým úkolem je vytvořit univerzální engine pro více her. Tím pádem jsem se tvorbou příběhu nijak důkladněji nezabýval. Pro můj účel stačilo pro každou hru vymyslet jednu či dvě věty. Mé příběhy zní následovně:

- Raketa plující vesmírem střílející z děla na náhodně generované nepřátele.
- Tank jedoucí nepřátelským územím snažící se zničit nepřátele.
- Autíčko jedoucí po cestě a vyhýbající se překážkám.
- Skákající panáček vyhýbající se překážkám.

Pokud je vymyšlen příběh, může se programátor pustit k implementaci vlastních her či enginu.

3.2 Vývojové prostředí

Ze všeho nejdříve je potřeba kvalitní, plně vybavené integrované vývojové prostředí (IDE). Veškerý kód pro tuto práci je psán v prostředí Eclipse, které je k dispozici bezplatně ke stažení. Vývojové prostředí Eclipse bylo zvoleno kvůli kvalitně zpracovaným návodům jak na internetu, tak v knihách. Mimo jiné i díky kvalitním zásuvným modulům, které pevně integrují spoustu jednotvárných ručních operací pro kompilování a ladění kódu pro systém Android.

3.3 Výběr verze platformy Android

Programátor má při tvorbě projektu možnost vybrat si ze všech verzí SDK, které jsou k dispozici. Pro tento engine byla jako nejnižší verze vybrána verze 2.2 (Froyo), která je stále na většině starších a výkonnostně slabších zařízeních. Maximální verze je průběžně aktualizována, aby si hry mohli zahrát hráči i s nejnovějšími telefony. Na začátku psaní práce to byla verze 4.1 (Jelly Bean). V průběhu byla aktualizována na nejnovější verzi 4.4.2 (KitKat).

3.4 OpenGL ES

Pravděpodobně jedním z nejdůležitějších prvků, který se používá při tvorbě her pro chytré telefony, je OpenGL ES, což je knihovna, kterou vyvinula společnost Silicon Graphics v roce 1992 pro použití v aplikacích typu CAD. Od té doby se o ni stará konsorcium Khronos Group a lze ji najít na většině platforem. Jedná se o velice výkonný a neocenitelný nástroj pro kohokoli, kdo chce vytvářet hry.

Verze Knihovny OpenGL, která je poskytována a podporována platformou Android, se ve skutečnosti označuje jako OpenGL ES. Knihovna OpenGL ES není tak bohatě vybavená jako standardní knihovna OpenGL. I tak se ovšem jedná o výjimečný nástroj pro vývoj softwaru na platformě Android.

Poznámka 3.1 V rámci této práce se na funkce a knihovny systému OpenGL ES budu pro jednoduchost odkazovat jako na OpenGL. Ve skutečnosti je používána OpenGL ES.

Většina lidí si knihovnu OpenGL ze všeho nejdříve spojí s trojrozměrnou grafikou. Faktem je, že knihovna OpenGL je velmi dobrá při vykreslování trojrozměrné grafiky a lze ji použít pro tvorbu přesvědčivých trojrozměrných her. Knihovna OpenGL je ovšem velice dobrá také při vykreslování dvojrozměrné grafiky. Ve skutečnosti dokáže vykreslovat a manipulovat s dvojrozměrnou grafikou mnohem rychleji než nativní volání systému Android. Nativní volání systému Android jsou pro většinu vývojářů aplikací dostatečná, avšak pro hry, které vyžadují maximální možnou míru optimalizace, je OpenGL tou nejlepší volbou.

Knihovna OpenGL je skvělý nástroj, protože jde o knihovnu pracující na více platformách. To znamená, že knihovnu OpenGL a znalosti pro její používání, které jsem se naučil, můžu používat v mnoha prostředích a oborech. Ve všech systémech, od iOS až po Microsoft Windows a Linux může být použito stejných funkcí knihovny OpenGL. Knihovna OpenGL nabízí užitečný, flexibilní a docela zavedený způsob pro práci s grafikou. Zpočátku byla implementace standardu OpenGL na systému Android velmi chybová a neobsahovala tolik funkcí, jako na jiných systémech. S příchodem dalších verzí systému Android se však implementace standardu OpenGL stále upevňovala.

3.4.1 Verze OpenGL ES

Pro uživatele platformy Andoid jsou k dispozici tyto verze knihovny OpenGL: OpenGL ES 1.0, 1.1, 2.0 a 3.0. V březnu roku 2014 byla zveřejněna i verze 3.1. Já si pro jednoduchost vybral verzi 1.0. Tato verze má k dispozici spoustu odborného materiálu a je také, mimo jiné, odzkoušená a otestovaná. Jakožto nejstarší z platforem OpenGL ES je dostupná na většině zařízení. Je i jednoduché s ní pracovat a přechod na novější verze je o to snazší.

3.5 Kód specifický pro hru

Jak bylo již zmíněno dříve, kód specifický pro hru je kód, který je pouze v jediné hře, na rozdíl od herního enginu, který lze sdílet a adaptovat pro různé hry. Kód specifický pro hru se skládá z veškerého kódu, který tvoří postavy ve hře (raketa, balon, nepřítel, přítel),

zatímco herní engine tyto postavy jen vykresluje. Kód specifický pro hru ví, že hlavní postava vystřelila z kanónu, zatímco herní engine danou věc vykreslí. Kód specifický pro hru zničí hlavní postavu, narazí-li na nepřátele. Herní engine jen testuje kolizi dvou objektů na obrazovce. Kolize dvou objektů by mohla v zjednodušeném úryvku vypadat následovně.

```
Hrac hrac;
Nepritel  nepritel ;
Nepritel  poleNepratel[] = new Nepritel[1];

poleNepratel[0] = nepritel ;

//presun postavy
PohybHrace(hrac);
PohybNepratel(poleNepratel);

// otestuj  kolizi
if (TestKolize(hrac, poleNepratel))
{
   Znic(hrac);
}
```

Výpis 1: Ukázka kolize dvou objektů

V tomto příkladu objekty *hrac* a *poleNepratel* a funkce *Znic()* náleží do kódu specifického pro hru, zatímco funkce *PohybHrace()*, *PohybNepratel()* a *TestKolize()* jsou součástí herního enginu. Na tomto krátkém příkladu lze snadno vidět, že místo objektu *hrac* a *poleNepratel* by mohlo být použito jakékoli jiné postavy v téměř jakékoli jiné hře a přitom by funkce *PohybHrace()* a *TestKolize()* i nadále pracovaly tak, jak mají.

3.6 Vlastní engine

Herní engine, který byl vytvořen pro tuto práci, je malinko odlišný od obecného herního enginu, který může být použit. Platforma Android je postavená na linuxovém jádře a vývoj softwaru se provádí pomocí mírně upravené verze Javy. To znamená, že Android je sám o sobě dostatečně rychlý k tomu, aby na něm snadno fungovaly některé příležitostné, malé hry. Toho jsem využil, aby programování nebylo příliš obtížné.

Nebyl sestaven skutečný, nízkoúrovňový engine jednoduše proto, že pro vytvořené hry nebyl nutný. Android obsahuje systémy, které mohou být využity, přestože nemusejí být optimální pro běh špičkových her. Snadno se však osvojují a hodí se pro druh her, které byly vytvořeny. Herní engine využívá sadu Android SDK (a související balíčky Javy) k provádění následujících činností:

- vykreslování grafiky,
- přehrávání zvuku a efektů,
- interpretování příkazů,
- detekce kolizí,

- obsluha umělé inteligence nepřátel,
- jednoduchá fyzika.

Díky jednoduchosti vytvořených her byl kód specifický pro hru a engine tvořen současně.

3.7 Práce s vlákny ve hře

Jednou z největších překážek, kterou musí vývojáři her překonat, je způsob funkčnosti her na libovolné platformě. V nejzákladnějším kořenovém prvku je hra pro platformu Android stále jen základní aktivitou systému Android. Každá další aplikace, která je napsána pro platformu Android, je také napsána jako aktivita. Jediným rozdílem mezi aktivitami je ten, že aktivita pro hru obsahuje hru, zatímco jiné mohou být obchodními, mapovými nebo sociálními nástroji. Problém tkví v tom, že vzhledem k tomu, že všechny aktivity Androidu jsou stejné, zachází se s nimi jako se stejnými. To znamená, že každá aktivita běží v hlavním prováděcím vláknu systému, což je pro hry neefektivní.

Běh hry v hlavním prováděcím vláknu systému znamená, že hra musí soupeřit o prostředky se všemi ostatními aktivitami běžícími v tomto vláknu. To vede k pomalé, trhané hře, která způsobí zastavení zařízení.

Pro obejití tohoto problému je možnost vytvořit libovolný počet vláken a spouštět v nich cokoliv, co chceme. V ideálním případě by měla hra být naprogramována v jednom vlákně a oddělena od všeho, co může být spuštěno na zařízení. To slouží pro plynulý běh hry a hra má přístup k těm prostředkům, které potřebuje. Ukázka znázorňuje vytvoření dvou samostatných vláken pro provádění hry. První vlákno pro běh hry a druhé vlákno pro běh podbarvující hudby, která hraje během hry.

```
/* spust uvodni obrazovku a hlavni nabidku v pozdrzenem vlaknu*/
 new Handler().postDelayed(new Thread() {
     @Override
     public void run() {
          Intent mainMenu =
               new Intent(MainActivity.this, MainMenu.class);
          MainActivity.this.startActivity(mainMenu);
          MainActivity.this.finish();
          overridePendingTransition(R.layout.fadein,R.layout.fadeout);

     }
 }, Engine.GAME_THREAD_DELAY);

/* spust podbarvujici hudbu */
 SFEngine.musicThread = new Thread() {
     public void run() {
          Intent bgmusic =
               new Intent(getApplicationContext(), Music.class);
          startService(bgmusic);
          Engine.context = getApplicationContext();
     }
 };
```

```
Engine.musicThread.start();
```

Výpis 2: Ukázka vytvoření vláken pro plynulý běh hry

3.8 Hudba ve hře

Hudba v enginu je naprogramována v samostatném vláknu sloužícímu pro přehrávání podbarvující hudby hrající na pozadí. Pro efektivitu byla zvolena hudba v podobě malého, 15 vteřinového cyklu, který se opakuje. Jsou-li soubory s médii příliš velké, mohou pro svou aktivitu spotřebovat veškerou dostupnou paměť, a způsobit tak její pád. Do enginu byly přidány konstanty, které se použijí v hudební službě.

```
public static final int SPLASH_SCREEN_MUSIC = R.raw.hudba;
public static final int R_VOLUME = 100;
public static final int L_VOLUME = 100;
public static final boolean LOOP_BACKGROUND_MUSIC = true;
```

Výpis 3: Ukázka konstant pro hudební službu v enginu

Konstanta *SPLASH_SCREEN_MUSIC* ukazuje na hudební soubor, který bude přehrávat, v tomto případě hudba.ogg. Konstanty *R_VOLUME* a *L_VOLUME* obsahují počáteční hlasitost pro hudbu a konstanta *LOOP_BACKGROUND_MUSIC* obsahuje pravdivostní hodnotu, která říká službě, zda se má hudba přehrávat cyklicky.

3.9 Vykreslování pozadí

Pozadí udává tón a prostředí hry. Pozadí je tvořeno různými prostředími v závislosti na typu hry. U všech je použita knihovna OpenGL pro vsazení pozadí do hry a jeho vykreslení na obrazovku. Tato knihovna slouží pro rychlé nakreslení pozadí hry. Pro tento engine bylo vytvořeno jedno i dvouvrstvé, opakující se, posuvné pozadí. Konkrétně větší obrázek, který se bude posouvat (a opakovat) a který je částečně překrytý druhým obrázkem posouvajícím se vyšší rychlostí (při dvouvrstvém pozadí).

Při vykreslování grafiky na platformě Android lze použít plátno (canvas). To ovšem nesplňuje podmínky flexibility, které jsou zapotřebí pro typ her, které tvořím. Knihovna OpenGL má svůj vlastní typ plátna, které je nutné použít pro vyobrazení grafiky na obrazovce. Zobrazení GLSurfaceView umožňuje zpodobnit herní grafiku na obrazovce.

Obrázky se v knihovně OpenGL nahrávají jako textury. Podstatné je, že jakýkoli obrázek, který chceme zobrazit pomocí knihovny OpenGL, je ve skutečnosti považován za texturu, která se aplikuje na trojrozměrný objekt. I když ve svých hrách vytvářím dvojrozměrnou grafiku, knihovna OpenGL s ní zachází jako s trojrozměrnými objekty.

3.9.1 Vykreslování povrchu a maticové režimy OpenGL

OpenGL obsahuje různé maticové režimy, které umožňují přistupovat k různým částem knihovny. Projekční matice poskytuje přístup ke způsobu, jakým se scéna vykresluje. Pro přístup k metodě *glOrthof()* musíme přepnout knihovnu OpenGL do režimu projekční

matice. Pro přístup k režimu projekční matice se musí zavolat metoda *glMatrixMode()* s argumentem *GL_PROJECTION*. Metoda *glOrthof()* připraví ortogonální, dvojrozměrné vykreslování scény. Tato metoda přijímá šest parametrů, z nichž každý definuje ořezávací rovinu.

Ořezávací roviny říkají vykreslovací jednotce, kde má zastavit vykreslování. Jinými slovy: jakékoli obrázky, které padnou mimo ořezávací roviny, budou při vykreslování zahozeny. Šest ořezávacích rovin jsou: levá, pravá, dolní, horní, blízká, vzdálená. Představují body na osách x, y a z.

Nastavením knihovny OpenGL do režimu texturové matice, nebo i do režimu modelové matice (používané pro posun a škálování vrcholů), získáme přístup ke všem texturám a všem vrcholům v OpenGL v daném okamžiku. To znamená, že když přepneme knihovnu OpengGL do režimu texturové matice a posuneme texturu o 1 jednotku na ose x, ve skutečnosti posouváme o jednu jednotku na ose x všechny textury, které máme v daný okamžik v OpenGL k dispozici.

Tato situace by mohla být problematická ve hře, kde bychom měli libovolný počet prvků posouvajících se a škálovaných v různých okamžicích různými rychlostmi a směry. Existuje logický způsob, jak vyřešit situaci posouvání jednotlivých prvků zvlášť, když knihovna OpenGL pracuje se všemi texturami a všemi vrcholy naráz.

Všechny maticové režimy se uchovávají na zásobníku. Jde o to, uložit matici na zásobník (v tomto případě texturovou matici) a poté posunout všechny textury a překreslit pouze ty, na které má mít daný pohyb vliv. Potom je třeba texturovou matici zase vyjmout ze zásobníku a celý proces opakovat pro další texturu, s níž se má pohnout.

Nesmí se zapomenout resetovat maticový režim zpět do jeho výchozího stavu před tím, než se s ním začne pracovat, jinak by měl totiž hodnotu, kterou měl naposled nastavenou. Resetování matice se provádí pomocí metody *glLoadIdentity()*.

3.9.2 Třída s pozadím

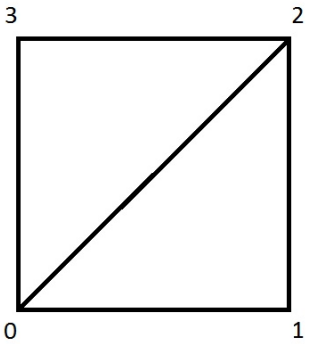
Třída s pozadím obsahuje pole pro uchovávání mapovacích souřadnic textury, pole pro uchovávání souřadnic vrcholů, pole pro uchovávání indexů vrcholů a pole ukazatelů na textury.

Pole *vertices[]* obsahuje řadu bodů. Každý řádek zde představuje hodnoty x, y a z rohu čtverce. V tomto případě je vytvořen čtverec, který zabírá celou obrazovku. Tím je zajištěno, že obrázek pokryje celou obrazovku.

Pole *texture[]* představuje místa, kde se rohy obrázku (neboli textury) spojí s rohy čtverce, který je vytvořen. V tomto případě textura pokrývá celý čtverec, který pokrývá celé pozadí.

Pole *indices[]* uchovává definici pro stěnu čtverce. Stěna čtverce je rozdělena na dva trojúhelníky. Hodnoty v tomto poli jsou rohy těchto trojúhelníků uvedené proti směru hodinových ručiček. Jedna úsečka (dva body) přesahuje (0 a 2). Obrázek 3.1 znázorňuje tuto koncepci.

Tyto tři druhy polí byly použity u každé třídy, která pracuje s texturami. Byly použity při vykreslování hráče, nepřátel, zbraní.



Obrázek 3.1: Označené indexové body [13]

3.9.3 Posouvání pozadí

Pokud by byl zavolán pouhý obrázek, obdrželi bychom statický obrázek pozadí. Statické pozadí je dobré pro hry, kde nepotřebujeme vytvořit například simulaci letu nebo jízdy autem. V první hře simulujeme let vesmírem a boj s nepřáteli, přičemž při simulaci letu vesmírem je nutné, aby se pozadí posouvalo.

V dřívější části byly zmíněny dva maticové režimy knihovny OpenGL: texturový a projekční. Pro posunutí textury na vrcholech je nutné přepnout knihovnu OpenGL do režimu texturové matice. V tomto případě neposouváme vrcholy, ale texturu na vrcholech. Vzhledem k tomu, že neposouváme vrcholy, musíme zajistit, aby byly na správném místě a nic se neúmyslně nepohnulo.

3.10 Běh rychlostí 60 snímků za vteřinu

Ideálem pro rychlost běhu her je 60 snímků za vteřinu. Pro plynulý herní zážitek by tedy měla hra běžet co nejblíže tomuto číslu. Tato problematika se dá vyřešit pozastavením vlákna, které zajistí plynulost rychlosti kolem 60 snímků za vteřinu.

Výhoda při použití vykreslovací jednotky GLSurfaceView, jakožto hlavního vstupního bodu hry, je to, že běží v samostatném vlákně. Pokud se explicitně neurčí jinak, bude se vykreslovací metoda volat nepřetržitě. Nemusí být vytvářena nová vlákna nebo volány metody hry v cyklu. Jakmile je nastavena instance vykreslovací třídy jako hlavní zobrazení aktivity, bude se v samostatném vláknu provádět akce, jež bude nepřetržitě volat vykreslovací metodu vykreslovací třídy.

Proto musíme běh této metody omezit a dohlédnout na to, aby neběžela rychleji než 60 snímků za vteřinu. Následující kód zobrazuje pozastavení vlákna, který na určitou dobu vlákno uspí. Dobou pro uspání vlákna bude jedna vteřina podělená číslem 60. Tato hodnota je uložena v konstantě, která je součástí třídy Engine a která má na starost všechny konstanty společné pro hru. Jedná se o nejvhodnější rychlost pro hry. Avšak této rychlosti nelze vždy dosáhnout. Čím více funkcí se v herním cyklu provádí, tím déle bude

dokončení cyklu trvat a tím pomaleji hra poběží. Proto je nutné prodlevu poupravit, nebo úplně vypnout podle toho, jak pomalu hra běží.

```
@Override
public void onDrawFrame(GL10 gl){
    //kod pro pozastaveni vlakna pro efekt 60 snimku za vterinu
    try    {
        Thread.sleep(SFEngine.GAME_THREAD_FPS_SLEEP);
            }
    catch (InterruptedException e) {
            e.printStackTrace();
            }
    //cokoli  za blokem try catch se bude spoustet 60x za vterinu
```

Výpis 4: Ukázka kódu pro pozastavení vlákna hry.

Cokoli, co bude nyní uvedeno v kódu za příkazem *try/catch* obsahující volání *Thread.sleep()*, se bude spouštět 60krát za vteřinu.

3.11 Tvorba objektů

Jedním z prastarých nástrojů při vývoji dvojrozměrných her je spritová animace. Při vzpomínce na oblíbené dvojrozměrné hry existuje velká šance, že animace kteréhokoli z objektů bylo dosaženo právě pomocí spritové animace.

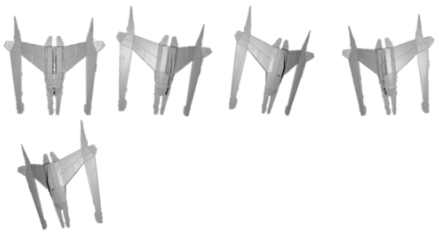
Sprite je v podstatě jakýkoliv grafický prvek v dvojrozměrné hře. Každý objekt v dané hře ze své podstaty spritem. Sprity jsou samy o sobě jen statické obrázky, které leží na obrazovce a nemění se. Spritová animace je proces, pomocí něhož lze vdechnout objektu jakýsi život, přestože tímto objektem je jen autíčko nebo vesmírná loď.

Spritové animace se dosahuje pomocí efektu rychle obrácených stránek knihy. Představme si téměř jakoukoli dvojrozměrnou hru, třeba Mario Brothers. Mario Brothers je jedním z nejlepších příkladů dvojrozměrných plošinových her, které obsahují spritovou animaci. V této hře pohybujeme Mariem doleva nebo doprava do strany se posunujícím prostředím. Mario jde nebo běží ve směru, kterým s ním pohybujeme. Jeho nohy jsou zřetelně animované v posloupnosti tvořící chůzi.

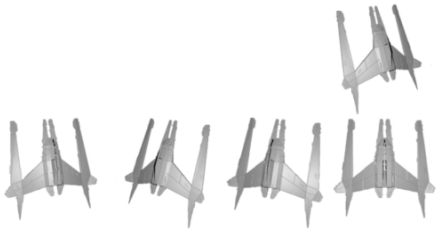
Tato animace chůze je ve skutečnosti tvořená posloupností statických obrázků. Každý znázorňuje různý okamžik chůze. Když hráč pohybuje postavou doleva nebo doprava, jednotlivé obrázky se vyměňují, což vytváří iluzi, že Mario jde. Při vytváření enginu byla využita stejná metoda pro vytváření animace pro postavy. Jednou z hlavních hratelných postav je vesmírná loď. Animace chůze zde nebylo potřeba. Vesmírné lodě ovšem nějakou animaci vyžadují. Byla vytvořena animace pro naklánění doprava a pro naklánění doleva podle směru, ve kterém hráč letí.

Textury používané v implementaci spritové animace nejsou z technického hlediska samostatné obrázky. Čas a výkon, který by byl zapotřebí pro nahrání a namapování nové textury 60krát za vteřinu (pokud by to bylo možné), by zdaleka překročil možnosti zařízení se systémem Android. Místo toho je použit arch spritů (sprite sheet). Arch spritů je jediným obrázek, který obsahuje všechny jednotlivé obrázky nutné k provádění spritové animace. Obrázek 3.2 ukazuje arch spritů pro hlavní hratelnou loď.

Poznámka 3.2 Knihovna OpenGL vykresluje veškeré bitmapy od posledního řádku k prvnímu. Tento arch spritů se tedy při vykreslování knihovnou OpenGL objeví na obrazovce nikoli jako obrázek 3.2, ale jako na obrázku 3.3



Obrázek 3.2: Arch spritů hlavní postavy



Obrázek 3.3: Jak se arch spritů vykreslí na obrazovce

Animace textury není tak složitá, jak se zdá. Obrázek se nahraje jako jedna textura. Zobrazovat se bude jen ta část textury, jež má obrázek, který se má zobrazit hráči. K animaci slouží metoda *glTanslateF()*, která přesouvá obrázek k další části, kterou chceme zobrazit. Při práci s vykreslováním postavy jsou použity oba již dříve zmíněné maticové režimy knihovny OpenGL. Modelová matice i texturová matice.

- Texturová matice je použita pro pohyb textury s archem spritů,
- modelová matice je použita pro pohyb postavy podél osy x či y.

Do enginu byly postupně přidány další objekty podle typu hry. Všechny objekty mají vlastní třídu a obsahují stejné vykreslovací metody knihovny OpenGL, tudíž jejich vykreslování je stejné. Tyto objekty se chovají buď staticky, nebo dle nastavené umělé inteligence. Objekty je možno ničit, nebo na ně jinak reagovat pomocí detekce kolizí.

3.12 Detekce kolizí a fyzika v enginu

Detekce kolizí určí, zda se dva objekty na obrazovce dotkly, a je základem pro jakoukoli videohru. V hrách se detekce kolizí používá k tomu, aby hráč nemohl procházet zdmi, mohl sbírat nové předměty, nebo se pomocí ní dokonce stanoví, zda nepřítel může ze zacloněného výhledu vidět hráče. Tato problematika souvisí s fyzikou, která je další důležitou vlastností enginu.

Aby hry vypadaly reálně, je zapotřebí implementovat základní fyziku. Když se řekne herní fyzika, vybaví se většině lidem destrukce a kolize, které samozřejmě značnou měrou přispívají k obrazu reálného světa. V reálném světě fungují veškeré činnosti podle obecně platných fyzikálních zákonů.

V tomto enginu byla fyzika a detekce kolizí naprogramována jednoduše. Automobil se o překážku zastaví, ale není počítáno se silami, které při nárazu působí na překážku nebo auto. Dalším příkladem mohou být válečné hry, kde výstřel z tanku znamená destrukci jako by tomu bylo v reálném světě.

3.13 Použití enginu

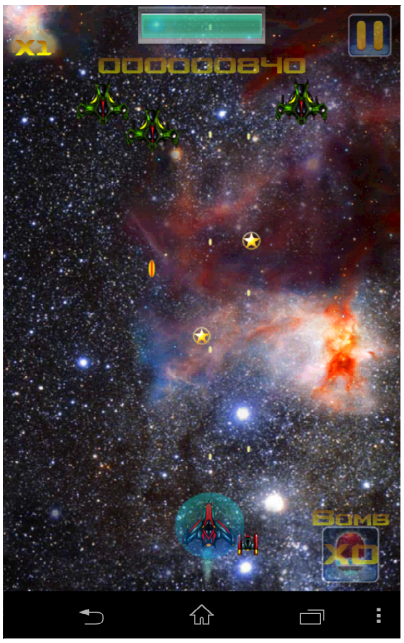
Engine je vytvořen pro jednoduché hry typu Super Mario, kde hráč putuje prostředím a snaží se vyhnout nepřátelům nebo je zničit. K těmto faktům bylo přihlíženo při tvorbě her.

4 Vytvoření ukázkových aplikací, srovnání s existujícími rámci

Postup vytváření ukázkových aplikací je znázorněn právě v této kapitole. Čtenář zde najde základní popis tvorby postav, jejich chování za použití umělé inteligence v případě nepřátel, či ovládání hlavní hrací postavy dotekem na displej. Je zde znázorněn postup tvorby základní detekce kolizí a fyziky v hrách.

4.1 Hra s raketou letící vesmírem

První hrou hra byla vytvořena raketa letící vesmírem a sestřelující náhodně generované nepřátele. Hra vznikla jako reakce na hru Shooter od studia Magma mobile [18]. Ukázková hra obsahuje pouze jeden level, ve kterém je 20 nepřátel. Obrázky 4.1 a 4.2 znázorňují ukázku ze hry Shooter a mnou vytvořenou hru.



Obrázek 4.1: Hra od herního studia Magma Mobile

Obrázek 4.2: Mnou naprogramovaná hra

4.1.1 Tvorba postavy a její pohybování pomocí události doteku

Nejdříve byla vytvořena hlavní hratelná postava. Tento objekt má stejné základní vykreslovací vlastnosti jako ostatní objekty, avšak neobsahuje žádnou umělou inteligenci, protože její pohyb a chování zajistí hráč sám.

Postava se pohybuje pomocí jednoduchého posluchače dotyků, které detekuje, zda se hráč dotkl levé nebo pravé strany obrazovky. Hráč pohybuje postavou doleva nebo doprava dotykem na příslušné straně obrazovky. Posluchač je definován pomocí metody *onTouchEvent()* v herním cyklu. Metoda *onTouchEvent()* je standardní posluchač událostí

systému Android, který naslouchá jakékoli události dotyku, k níž dochází v rámci dané aktivity. Posluchač *onTouchEvent()* se spustí pouze tehdy, dojde-li k dotyku, úderu, tažení nebo uvolnění na obrazovce zařízení. U této hry nás zajímá pouze dotyk nebo uvolnění a na které straně obrazovky k tomu došlo. Abychom to mohli určit, posílá systém Android do posluchače *onTouchEvent()* objekt typu *MotionEvent*, který obsahuje vše, co potřebujeme pro stanovení, o jaký druh události dotyku se jedná a kde k dotyku na obrazovce došlo.

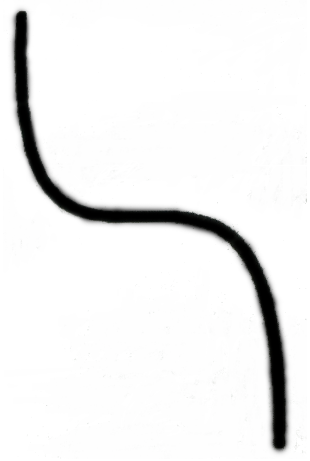
Objekt typu MotionEvent má velmi užitečnou metodu s názvem *getAction()*, která vrací typ akce, jež byla detekována na obrazovce. Pro účel této hry byly zvoleny akce *ACTION_UP* a *ACTION_DOWN*. Tyto akce signalizují okamžiky, kdy se prst hráče zpočátku dotkl obrazovky a potom se zvedl z obrazovky pryč.

4.1.2 Přidání nepřátel

Ve hře s vesmírnou lodí byly vytvořeny tři druhy nepřátel. Pro jednoduchost jsem je pojmenoval jako stíhač, průzkumník a válečná loď. Třída nepřátel má stejné chování jako třída s hlavní postavou hráče. V případě této hry byla přidána umělá inteligence pro nepřátele pro větší požitek ze hry. Dva typy nepřátel, které vytvářím, se pohybují v diagonálních, avšak přímých směrech. Jeden nepřítel se pohybuje způsobem známým jako Bézierova křivka.

4.1.3 Bézierova křivka

K tomu, aby nepřítel letěl po kvadratické Bézierově křivce od horní k dolní části obrazovky, jsou zapotřebí dvě metody: jedna pro následné x-ové pozice na Bézierově křivce a jednu pro získání následné y-ové pozice na Bézierově křivce. Při každém volání této metody je obdržen následující bod na osách x a y, na který je zapotřebí daného nepřítele přesunout. Obrázek 4.3 ukazuje, jak Bézierova křivka vypadá.



Obrázek 4.3: Kvadratická Bézierova křivka

Zakreslování bodů není naštěstí nijak složité. Pro sestrojení kvadratické Bézierovy křivky je zapotřebí čtyř bodů v kartézské soustavě souřadnic: začátek, konec a dva body někde mezi nimi, kolem nichž se bude křivka otáčet. Tyto body se ve hře nikdy nezmění. Každý nepřítel bude sledovat stejnou křivku a to buď zleva, či zprava. Proto bylo v Enginu vytvořeno osm konstant reprezentující čtyři body kvadratické Bézierovy křivky na osách x a y.

Klíčovou hodnotou při zakreslování těchto bodů je koeficient t. Koeficient t říká vzorci, kde se na křivce objekt nachází, díky čemuž může vzorec vypočítat souřadnici x nebo y pro tuto pozici. Zde je vzorec pro nalezení bodu na kvadratické Bézierově křivce na ose y (pro nalezení hodnoty na ose x se zamění y za x):

$$(y1 * t3) + (y2 * 3 * t2 * (1 - t)) + (y3 * 3 * t(1 - t)2) + (y4 * (1 - t)3) \tag{4.1}$$

Zde je názorná ukázka použití vzorce pro výpočet kvadratické Bézierovy křivky v kódu.

```
public float getNextBezierY(){
    return (float) ((Engine.BEZIER_Y_1*(posT*posT*posT)) +
        (Engine.BEZIER_Y_2 * 3 * (posT * posT) * (1−posT)) +
        (Engine.BEZIER_Y_3 * 3 * posT * ((1−posT) * (1−posT))) +
        (Engine.BEZIER_Y_4 * ((1−posT) * (1−posT) * (1−posT))));
}
```

Výpis 5: Ukázka kódu pro výpočet kvadratické Bézierovy křivky

4.1.4 Umělá inteligence nepřátel

Umělá inteligence nepřítele definuje, jak nepřítel útočí na hráče a jak je pro hráče snadné nebo obtížné ve hře zvítězit. Proto byly přidány tři odlišné umělé inteligence pro tři odlišné typy nepřátel. Vytváření nepřátel se může zdát jako lehký úkol, skutečnost je však taková, že vytváření nepřátel je obtížnější než tvorba hratelné postavy. Hratelné postavy nemusejí přemýšlet - dělají jen to, co jim hráč přikáže. Nepřátelé naproti tomu musejí mít alespoň základní umělou inteligenci, která je během hry povede.

První krok pro přidání umělé inteligence nepřátel spočívá ve vytvoření metody *pohybNepratel()*, která obsahuje veškerou logiku pro nepřátele. Metoda *pohybNepratel()* se volá v herním cyklu pro aktualizaci pozic nepřátelských lodí. Metoda aktualizuje v jednom volání všechny nepřátele. Zpracování všech nepřátel v jednom volání je vždy nejlepší způsob, jak přistupovat k aktualizaci velkého množství nehratelných postav. Ušetří to drahocenné cykly procesoru.

Stíhač se začne pohybovat přímočaře dolů podél osy y. V určitém bodě na cestě podél osy y se zaměří na loď hráče a letí přímo na tyto souřadnice ve snaze do této lodě narazit.

Ze všeho nejdříve se musí otestovat, zda stíhač je, či není mimo obrazovku. Všichni nepřátelé se pohybují po obrazovce shora dolů, a pokud je nezničí hráč, nakonec dosáhnout spodní části obrazovky. Při takovémto návrhu hry jsou dvě možnosti. Když nepřítel dosáhne spodní části obrazovky, můžeme jej buď zabít, nebo jej resetovat a nechat letět

znovu. Já se rozhodl pro resetování nepřátele na náhodnou pozici nad horní částí obrazovky, aby tak pokračovali v útoku na hráče, dokud nebudou zničeni. Tento problém byl vyřešen otestováním, zda je y-ová pozice nepřátele menší než 0 (tj. pod spodním okrajem obrazovky), a pokud ano, resetuje se jeho x-ová a y-ová souřadnice na náhodnou pozici.

Než se stíhač zaměří na pozici hráče, bude jednoduše cestovat dolů po obrazovce po přímé dráze. Vybral jsem libovolnou pozici na ose y. Jde o bod, na němž se stíhač zaměří na pozici hráče. V této hře je y-ová pozice stíhače v okamžiku, kdy se zaměří na hráče, na bodu 3, což znamená, že stíhač začne na náhodné pozici na ose y a pohybuje se dolů po přímé dráze, dokud nedorazí do bodu 3. Toho jsem docílil odečtením hodnoty rychlosti stíhače od aktuální y-ové pozice stíhače. Dále byl použit jednoduchý vzorec pro sklon pro stanovení přírůstku, podle nichž se má stíhač pohybovat, aby pak dorazil k hráči. Sklon lze určit podle následujícího vzorce:

$$(x1 - x2)/(y1 - y2) \tag{4.2}$$

V tomto vzorci byla nahrazena konstanta y2 konstantou rychlosti stíhače pro zrychlení nepřítele po zaměření se na cíl. Výše uvedený vzorec přenese stíhače přímo k hráči v jediném skoku. Stíhač se ovšem musí pohybovat k hráči v plynulých přírůstcích. Proto bylo provedeno místo odečtení y2 od y1 dělení y1 hodnotou y2. Tím byla získána inkrementální hodnota, která může být pro pohyb stíhače sčítána s ní samotnou. Následující kód zobrazuje vytvoření celé umělé inteligence stíhače.

```
private void pohybNepratel(GL10 gl){
   for(int x = 0; x < Engine.TOTAL_INTERCEPTORS + Engine.TOTAL_SCOUTS + Engine.
        TOTAL_WARSHIPS; x++){
      if (!nepratele[x].isDestroyed){
         Random randomPos = new Random();

         switch (nepratele[x].enemyType) {
         case Engine.TYPE_INTERCEPTOR:
            if (nepratele[x].posY <= 0) {
               nepratele[x].posY = randomPos.nextFloat() * 4 + 4;
               nepratele[x].posX = randomPos.nextFloat() * 3;
               nepratele[x].isLockedOn = false;
               nepratele[x].lockOnPosX = 0;
                 }
            //nastaveni velikosti nepratele
            gl.glMatrixMode(GL10.GL_MODELVIEW);
                 gl.glLoadIdentity();
                 gl.glPushMatrix();
                 gl.glScalef(.25f, .25f, 1f);

                 //zamereni na pozici hrace
                 if (nepratele[x].posY >= 3) {
                  nepratele[x].posY -= Engine.INTERCEPTOR_SPEED;
                 } else {
                  if (!nepratele[x].isLockedOn) {
                     nepratele[x].lockOnPosX = Engine.playerAktualPosX;
                     nepratele[x].isLockedOn = true;
```

```
            nepratele[x].incrementXToTarget = (float)((nepratele[x].lockOnPosX -
                nepratele[x].posX) / (nepratele[x].posY / (Engine.
                INTERCEPTOR_SPEED * 4)));
            }
        //nastaveni x-ove a y-ove pozice stihace
        nepratele[x].posY -=(Engine.INTERCEPTOR_SPEED * 4);
        nepratele[x].posX += nepratele[x].incrementXToTarget;

        //posunuti vrcholu podle nove x-ove a y-ove pozice stihace
        gl.glTranslatef(nepratele[x].posX, nepratele[x].posY, 0f);
        gl.glMatrixMode(GL10.GL_TEXTURE);
        gl.glLoadIdentity();
        gl.glTranslatef(0.25f, .25f , 0.0f);

        //nakresleni stihace
        nepratele[x].draw(gl, spriteSheets);
        gl.glPopMatrix();
        gl.glLoadIdentity();
    }

break;
```

Výpis 6: Umělá inteligence stíhače

Jediný podstatný rozdíl mezi průzkumníkem a stíhačem spočívá v tom, že průzkumník se bude pohybovat dolů po obrazovce předem definovaným způsobem. Průzkumník bude útočit pouze z krajní levé nebo krajní pravé strany obrazovky. Proto byla nastavena x-ová pozice buď na 0, nebo na 3, podle směru jeho útoku.

Válečná loď se pohybuje v náhodném směru. Toho bylo docíleno výběrem náhodné pozice na ose x a pohybem válečné lodě pomocí stejné logiky, jako má stíhač, směrem k tomu náhodnému bodu, a ne přímo na pozici hráče. Umělá inteligence pro válečnou loď bude téměř identická s umělou inteligencí stíhače, s výjimkou toho, že x-ová pozice zaměření hráče byla nahrazena náhodným číslem mezi 0 a 3.

4.1.5 Přidání zbraní hráči

Pro zbraně a kolize jsem zvolil nový arch spritů, abych si zkusil práci s více texturami najednou. Teoreticky by bylo možné začlenit zbraně do téhož archu spritů, na němž je hráč i nepřátelé. Pro vykreslení zbraně na obrazovku je zapotřebí znát tři věci: x-ovou a y-ovou pozici vrcholu spritu a je-li sprite právě na obrazovce. Pozice pomůže umístit sprite na správné místo na obrazovce a zjistit, zda nedošlo ke kolizím. Na obrazovce může být v jednu chvíli více střel, stejně jako více nepřátel. Proto je nutné mít zbraně v poli. Při procházení pole je potřeba vědět, zda je daná střela na obrazovce nebo je připravená na vystřelení.

Zbraň hráče střílí zcela automaticky. Proto metoda, která je vytvořena, střílí bez povelu hráče. Vzhledem k tomu, že střílení probíhá automaticky, je první střela nastavena jako právě vystřelená. Pozice střely na ose x odpovídá aktuální x-ové pozici postavy hráče.

Co se týče pozice na ose y, byla nastavena na hodnotu *1,25f*. Tím byla umístěna mírně nad loď hráče, díky čemuž vypadá, jako by vycházela z čelního děla. Pokud by byla y-ová

hodnota nastavena níže, střela by se nakreslila přes loď. Dráha každé střely je přímá čára, která se pohybuje od x-ové pozice hráče v době výstřelu po horní okraj obrazovky. Pro pohyb každé vystřelené střely po přímé dráze byla vytvořena metoda *strelaHrace()*.

V metodě je vytvořen cyklus, který zpracovává všechny vystřelené střely. Díky tomu se nezdržuje střelami, které není nutné kreslit. První věcí, která byla při zpracování střely učiněna, bylo vytvoření proměnné typu int s názvem *dalsiStrela*. Automatická střelba postavy hráče střílí jednotlivé střely v určitém sledu. Další střela by se proto neměla vystřelit, dokud předchozí střela neurazí přijatelnou vzdálenost od postavy. Proměnná *dalsiStrela* uchovává střelu, která se má následně vystřelit, aby tak mohlo být nastaveno po správném okamžiku několik počátečních vlastností.

Jednou z vlastností je detekování okraje obrazovky. Je potřeba způsob zjištění, zda střela zasáhla okraj pozorovatelného obrazu, aby se tak nekreslila v místě, kde ji hráč již nemůže vidět, což by vedlo k mrhání cennými prostředky. Toto bylo ošetřeno příkazem if, který testuje, zda je aktuální střela mimo obrazovku. Pokud se střela dostala mimo obrazovku, byla nastavena její vlastnost vystreleno na hodnotu false, čímž bylo zamezeno jejímu zbytečnému vykreslování.

Pokud střela dosud nepřekročila hranice obrazovky, musí být stále v zorném poli hráče a je nutné se jí zabývat. Střely letí po přímé dráze, a proto pro pohyb střely stačí k její aktuální y-ové pozici stále přičítat konstantu rychlosti letu střely. Poté se pracuje se všemi operacemi knihovny OpenGL pro vykreslení, s nimiž se pracovalo již při kreslení postavy na obrazovku.

V této metodě musí být ošetřena ještě jedna věc. Jakmile se aktuální střela posune po ose y o více než 1 jednotku od postavy, nastal čas na vystřelení další střely. Proto musí být otestováno, zda je aktuální střela vzdálena více než 1 jednotku na ose y od postavy, a pokud ano, nastavit vlastnosti pro střelu, která se má následně vystřelit. Střely se střídají postupně, takže v okamžiku vystřelení poslední střely by první střela již měla být mimo obrazovku a deaktivována. Jakmile tedy poslední střela opustí obrazovku, je možné první střelu znovu vystřelit. K tomu, aby byly střely účinné, je zapotřebí vytvořit detekci kolizí.

4.1.6 Detekce kolizí

V této hře je použita základní detekce kolizí pro ničení nepřátel. Detekce kolizí byla vyřešena pomocí metody, jež sleduje pozici každého z nepřátel na obrazovce a každé vystřelené střely pro stanovení, zda některá ze střel nezasáhla některého z nepřátel. Ve dvojrozměrné hře je tento proces snazší, protože stačí testovat pouze na dvou osách (nemusíme testovat osu z). Při detekování kolizí dochází také k znázornění fyziky v dané hře. Jak bylo zmíněno v předchozí kapitole, po nárazu střely do nepřátelské lodi bude tato loď zničena.

4.1.7 Tvorba metody detekce kolizí

Uvnitř metody *detekceKolizi()* se nachází dva cykly, jeden pro procházení všech střel a druhý pro procházení všech nepřátel, kteří ještě nebyli zničeni. Nepřátelé mohou být

kvůli své počáteční pozici nad horním okrajem obrazovky v platném stavu (*isDestroyed* == *false*), aniž by se nacházeli v zorném poli hráče. To znamená, že kromě příznaku zničení nepřítele musí být otestováno také to, je-li v zorném poli hráče.

Samotné detekování kolize bylo vyřešeno příkazem if, který zjišťuje, zda došlo ke kolizi střely a nepřítele, na základě jejich x-ových a y-ových pozic a příslušných rozměrů. Pokud testovaný nepřítel a střela projdou tímto příkazem if, pak došlo k jejich kolizi. V takové situaci stačí na daném nepříteli zavolat metodu *applyDamage()*, která buď zvýší jeho poškození, nebo jej zcela zničí. Jakmile střela zasáhne nepřítele, je nutné bez ohledu na to, zda je nepřítel zcela zničen, odebrat střelu z obrazovky, aby již nemohla zasáhnout žádné jiné nepřátele. Zde je uvedena ukázka kódu detekce kolizí implementované v této hře. Čtenář může porovnat s kódem uvedeným jako ukázku na začátku předchozí kapitoly.

```
private void detekceKolizi(){
    for(int y = 0; y < 3; y++){
        if (strela[y].vystreleno){
            for (int x = 0; x < Engine.TOTAL_INTERCEPTORS
                    + Engine.TOTAL_SCOUTS
                    + Engine.TOTAL_WARSHIPS; x++){
                if (!nepratele[x].isDestroyed && nepratele[x].posY < 4.25){
                    if ( (     strela[y].posY > nepratele[x].posY - 1
                            && strela[y].posY <= nepratele[x].posY)
                            && (strela[y].posX <= nepratele[x].posX + 1
                            && strela[y].posX >= nepratele[x].posX - 1)){
                        int dalsiStrela = 0;
                        nepratele[x].applyDamage();
                        strela[y].vystreleno = false;
                        if(y == 3){ dalsiStrela = 0;}
                        else { dalsiStrela = y + 1; }
                        if (strela[dalsiStrela].vystreleno == false){
                            strela[dalsiStrela].vystreleno = true;
                            strela[dalsiStrela].posX = Engine.playerAktualPosX;
                            strela[dalsiStrela].posY = 1.25f;
                        }
                    }
                }
            }
        }
    }
}
```

Výpis 7: Detekce kolize střely hráče a nepřítele

4.2 Ukázka hry Tank

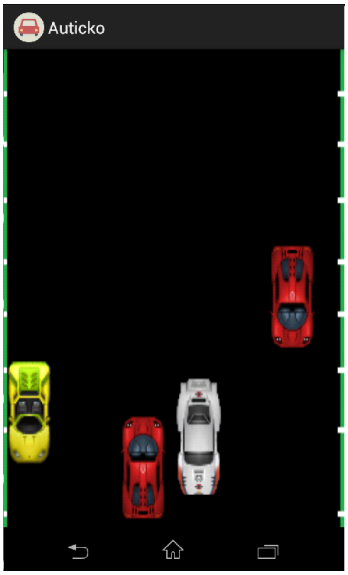
Další hru, kterou jsem se rozhodl vytvořit, je tank. Zde jsem chtěl demonstrovat, že pouhou výměnou textur a jednoduchým přeprogramováním rychlosti střely a rychlosti pohybu postav, lze vytvořit zcela novou hru. Umělá inteligence u nepřátel a pohyb hlavní hratelné postavy zůstaly stejné. Hra také obsahuje pouze jeden level, ve kterém je 25 nepřátel. Obrázek 4.4 znázorňuje druhou hru.

Obrázek 4.4: Ukázka druhé hry - tank

4.3 Ukázka hry Autíčko

Třetí hrou je autíčko jedoucí po cestě a vyhýbající se jiným autům. Inspirace pochází ze hry Somaliland Road Race od vývojáře Mataan [19]. Obrázek 4.5 ukazuje hru Somaliland Road Race a obrázek 4.6 mnou naprogramovaného autíčka.

Obrázek 4.5: Hra od vývojáře Mataan



Obrázek 4.6: Mnou naprogramovaná hra

V této hře je chování hlavní postavy stejné jako v předchozích dvou hrách. Hráč ovládá postavu dotekem na té straně displeje, na kterou chce autíčko přesunout. Hratelná plocha byla také nastavena na spodní čtvrtinu displeje. Bylo však pozměněno chování

nepřátel. Nepřátelská auta se pohybují shora dolů. Jejich počáteční pozice je náhodně vygenerována.

Detekce kolizí zde byla řešena tak, že jakmile nepřítelova y-ová pozice je stejná nebo menší jako y-ová pozice hráče a zároveň jsou i stejné jejich x-ové pozice v závislosti na rozměru auta, dojde ke kolizi a hra se ukončí. Tato detekce kolizí znázorňuje jednoduchou fyziku z reálného světa, kde při střetnutí auta s překážkou se auto zastaví.

4.4 Ukázka hry s panáčkem na vesmírném skateboardu

Poslední hrou, která byla naprogramována na mnou vytvořený jednoduchý 2D engine, je panáček na skateboardu. Hrací postava se ovládá jednoduše. Stačí zmáčknout kdekoli na displej a hráč vyskočí do vzduchu. Po sejmutí prstu z displeje začne panáček padat zpátky na zem. Těmito jednoduchými úkony se snaží hráč vyhnout překážkám, které se mu dostávají do cesty. Při nárazu do překážky se hra ukončí.

Chování hráče je zde simulováno jednoduchou gravitací, která je ovlivněna jeho vesmírným skateboardem, který dokáže létat. Při stisku displeje panáček vzlétne podle jednoduchého vzorce. K jeho aktuální y-ové pozici se přičte rychlost jeho pohybu a touto rychlostí stoupá do té doby, než hráč nespustí prst z displeje, nebo dokud nenarazí na vrchní okraj obrazovky. Při pádu zpátky na zem je od jeho aktuální y-ové pozice odečtena jeho rychlost a proto padá stejnou rychlostí, jakou vzlétnul.

Detekování kolizí bylo provedeno podobně jako v případě autíčka. Při doteku s nepřítelem nemůže panáček pokračovat v pohybu a hra je ukončena. Na obrázku 4.7 lze vidět ukázku ze hry.

Obrázek 4.7: Ukázka ze hry s panáčkem na létajícím vesmírném skateboardu

4.5 Testování a ladění her

Jako hlavní problém při testování aplikací pro platformu Android lze uvést velkou rozdílnost v rozlišení displejů a jejich velikostí. Programátor musí počítat s tím, že jeho aplikace si může uživatel zobrazit na telefonech s malými displeji, ale i na tabletech.

Mezi vývojové nástroje, které se používají při vývoji aplikací pro Android, neodmyslitelně patří emulátor zařízení - speciální software, který simuluje zařízení Android. Tento nástroj je při vývoji velmi užitečný, protože umožňuje nejenom zahájit vývoj pro Android bez fyzického zařízení, ale také umožňuje testovat konfigurace zařízení, která nemá programátor k dispozici.

Emulátor vestavěný do prostředí Eclipse je pomalý a hry se pomocí něj nemohou moc dobře testovat. Jako alternativu lze použít emulátor od firmy Genymotion, který je rychlejší a běžící zvlášť jako virtuální počítač pomocí programu Virtual Box [20].

Při tvorbě byly hry testovány na testovacím zařízení Sony Xperia T, kde fungovaly bez problémů a jejich běh byl plynulý. Výhodou byla okamžitá odezva a testování na reálném zařízení ušetřilo spoustu času. Dalším zařízením byl Tablet Sony Xperia Z, který byl simulován jako emulátor firmy Genymotion. Zde byla odezva rychlejší než u emulátoru, který poskytuje vývojové prostředí Eclipse. Testování na tomto zařízení proběhlo především proto, abych viděl, jak hra vypadá na tabletu, který nemám k dispozici.

Závěr

Cílem bakalářské práce bylo vytvořit vlastní jednoduchý herní engine pro mobilní platformu Android. Úvodní kapitola byla věnována samotné problematice Androidu, kde se čtenář dozvěděl základní informace z historie a současnosti, ale také zde nalezl základní stavební vlastnosti systému. Následující kapitola obnášela informace o již vytvořených a funkčních enginech pro tvorbu her a jejich srovnání. Hlavní část této práce byla samotná tvorba enginu. Byla zde použita grafická knihovna OpenGL ES, se kterou jsem se dříve nesetkal a naučil se s ní pracovat. Výstupem této práce je jak jednoduchý engine, tak pomocí něj vytvořené jednoduché 2D hry, které splňují zadání práce, a to reakce objektů vzájemně na sebe použitím základní detekce kolizí a jednoduché fyziky v enginu. Většina publikací na toto téma je v anglickém jazyce, proto jako jeden z přínosů vidím v tom, že jsem se naučil spousty nových anglických odborných slov. Navíc má čtenář této práce možnost informace získat v českém jazyce.

Ovšem jako největší zisk do mého osobního života vidím v tom, že jsem se díky této práci rozhodl, čím se chci i nadále zabývat. Svět platformy Android mě natolik zaujal, že jsem začal tvořit aplikace i ve svém volném čase. Nejedná se přímo o hry, ale spíše o aplikace pro každodenní využití.

Při vývoji her mě napadala možná vylepšení pro mnou vytvořené hry. Jednalo by se o hry s více úrovněmi obtížností, kde by si hráč mohl na začátku každé hry zvolit, jak obtížné pro něj bude dostat se k cíli nebo nejlepším výsledkům. Další rozšíření by se mohlo týkat vylepšení hrací postavy. Například raketa by mohla přikupovat různá nová děla, autíčko by mohlo jet rychleji apod. Jedno z vylepšení by se mohlo týkat i získávání bodů při počtu přeskočených překážek a soutěžit tak s jinými hráči o co nejvyšší skóre. Při dalších rozšířeních by mohly vzniknout plnohodnotné hry, které by se možná dočkaly úspěchu při publikování na Google Play.

Michal Zich

Literatura

[1] ALLEN, Grant. *Android 4 Průvodce programováním mobilních aplikací.* Brno: Computer Press, 2013. str. 656. ISBN: 978-80-251-3782-6

[2] E-SVET.E15.CZ. *Zelený robot dominuje: Android má na trhu chytrých telefonů podíl 79 procent* [online]. 2014 [cit. 2014-03-29]. Dostupné z: http://e-svet.e15.cz/technika/zeleny-robot-dominuje-android-ma-na-trhu-chytrych-telefonu-podil-79-procent-1058541

[3] DEVELOPER.ANDROID.COM. *Dashboards* [online]. 2014 [cit. 2014-03-27]. Dostupné z: http://developer.android.com/about/dashboards/index.html

[4] SVĚT ANDROIDA. *Alternativy ke Google Play – kde a jak sehnat zajímavé aplikace* [online]. 2014 [cit. 25. 3. 2014]. Dostupné z: http://www.svetandroida.cz/alternativy-ke-google-play-kde-sehnat-zajimave-aplikace-201403

[5] KRUMNIKL, Michal. *Livecycle of Android Activity* [online]. 2013 [cit: 2. 12. 2013] Dostupné z: http://tamz2.mrl.cz/download/TAMZ2-2012-2.pdf

[6] UNREAL ENGINE. *UDK Licensing* [online]. 2014 [cit. 29. 3. 2014]. Dotupné z: https://www.unrealengine.com/products/udk/udk-licensing-resources

[7] MOBILIZUJEME. *Unreal Engine 3 na Android zařízení? Přesvědčte se sami s aplikací Epic Citadel* [online]. 2014 [cit. 29. 3. 2014]. Dostupné z: http://mobilizujeme.cz/clanky/unreal-engine-3-na-android-zarizeni-presvedcte-se-sami-s-aplikaci-epic-citadel/

[8] UNITY3D, *www.unity3d.com* [online]. 2014 [cit. 29. 3. 2014]. Dostupné z: http://unity3d.com/unity/download

[9] ANDENGINE, *www.andengine.org* [online]. 2014 [cit. 29. 3. 2014]. Dostupné z: http://www.andengine.org/blog/

[10] CORONA SDK, *coronalabs.com* [online]. 2014 [cit. 29. 3. 2014]. Dostupné z: http://coronalabs.com/products/corona-sdk/

[11] COCOS2D-X, *www.cocos2d-x.org* [online]. 2014 [cit. 29. 3. 2014]. Dostupné z: http://www.cocos2d-x.org/wiki

[12] LIBGDX, *www.libgdx.badlogicgames.com* [online]. 2014 [cit. 29. 3. 2014]. Dostupné z: http://libgdx.badlogicgames.com/index.html

[13] ZECHNER, Mario. *Beginning Android Games.* New York: Apress, 2011. str. 669. ISBN: 978-1-4302-3042-7

[14] SILVA, Vladimir. *Pro Android Games. Second edition.* New York: Apress, 2012. str. 392. ISBN: 978-1-4302-4797-5

[15] DIMARZIO, J.F . *Programujeme hry pro Android 4.* Brno: Computer Press, 2012. str. 310. ISBN: 978-80-251-3751-3

[16] HERODEK, Martin. *Android Jednoduše.* Brno: Computer Press, 2013. str. 128. ISBN: 978-80-251-4118-2

[17] FRIESEN, Jeff. *Learn Java for Android Development, Second Edition.* New York: Apress, 2013. str. 780. ISBN: 978-1-4302-5722-6

[18] MAGMAMOBILE. *Magma Mobile Your Joyful Escape* [online]. 2014 [cit. 3. 4. 2014]. Dostupné z: http://www.magmamobile.com/

[19] MATAAN. *Mataan* [online]. 2014 [cit. 3. 4. 2014]. Dostupné z: http://www.mataan.com/

[20] GENYMOTION. *The faster Android emulator* [online]. 2014 [cit. 3. 4. 2014]. Dostupné z: http://www.genymotion.com/